FABRAI

ALGORIMTOS
E
TÉCNICAS
DE
PROGRAMAÇÃO

PROFa. Juliana Santiago Teixeira

Índice

1. O Computador: Hardware/Software

1.1. Hardware

1.1.1. Introdução

O computador eletrônico é um recurso formado por duas partes: uma porção **física** e uma porção **lógica**.

O nome técnico que se dá à porção física do computador é *hardware* e a sua porção lógica é *software*.

COMPUTADOR = HARDWARE +SOFTWARE
(física) (lógica)

1.1.2. Representação das Informações

Toda informação introduzida em um computador precisa ser entendida pela máquina.

O computador digital trabalha com lógica binária. Nesta lógica existem dois estados: circuito ligado ou desligado, passando ou não corrente elétrica, etc.

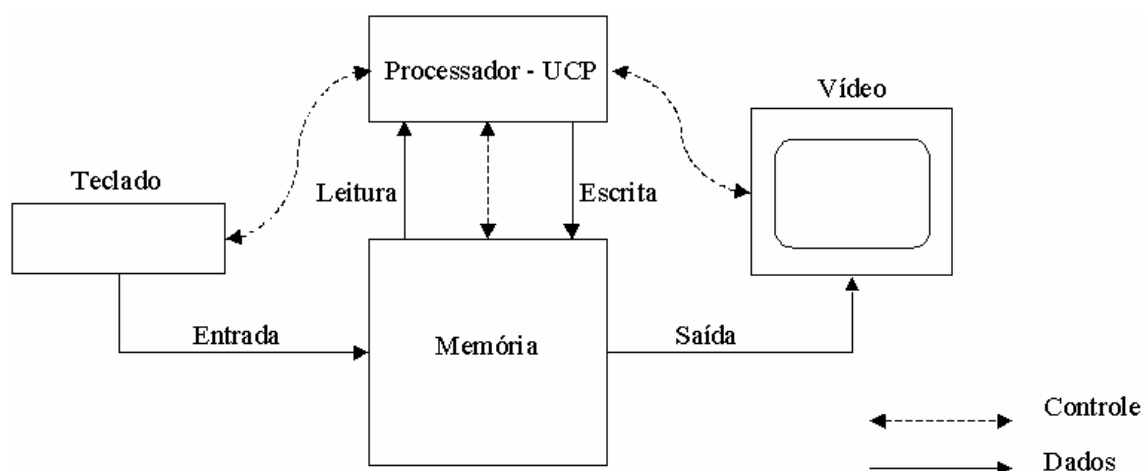
A menor unidade de informação armazenável em um computador é o algarismo binário ou dígito binário, conhecido como **bit** (**bi**nary digi**t**). O *bit* pode ter, então, somente dois valores: 0 e 1. O menor grupo ordenado de *bits* representando uma informação útil e inteligível para o ser humano é o **caractere**.

Outro termo bastante comum da área de informática é o **byte**. *Byte* é um conjunto de *bits*, usualmente oito *bits*. O *byte*, normalmente representa uma posição no computador e seu conteúdo pode caracterizar um algarismo, uma letra, um código, um símbolo, etc. Essa unidade (*byte*) é utilizada extensamente como unidade-padrão de medida das mais variadas capacidades em se tratando de computadores. Em termos de escala, existem os múltiplos do *byte*, e os mais utilizados são os seguintes:

Byte (B)……………	1 B…………..	1b
Kilobyte (KB).........	1024 B……...	2^{10}B
Megabyte (MB)…..	1024 KB…….	2^{20}B
Gigabyte (GB).......	1024 MB……	2^{30}B
Terabyte (TB)…….	1024 GB…...	2^{40}B
Petabyte (PB)......	1024 TB.......	2^{50}B

Além o *bit* e do *byte*, há um outro conceito, relacionado à forma com que os dados binários serão acessados e processados. Trata-se do conceito **palavra**. Se tomarmos um computador de oito *bits* para analisar, veremos que se trata de um equipamento cujo tamanho da *palavra* é de oitos *bits*. Isso significa que os dados binários são acessados e processados em conjuntos ou grupos de oito.

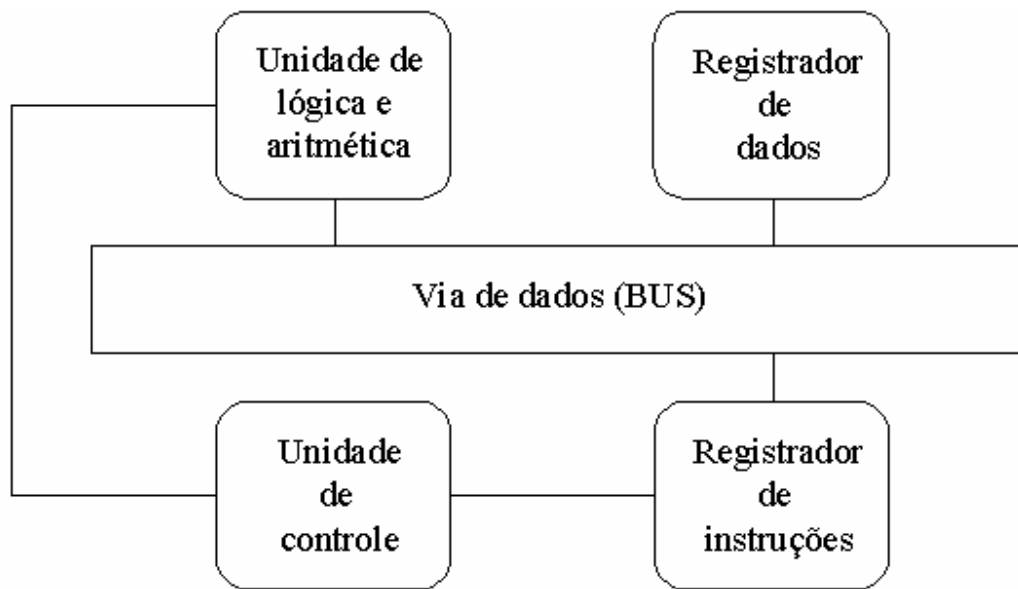
1.1.3. Componentes do Hardware

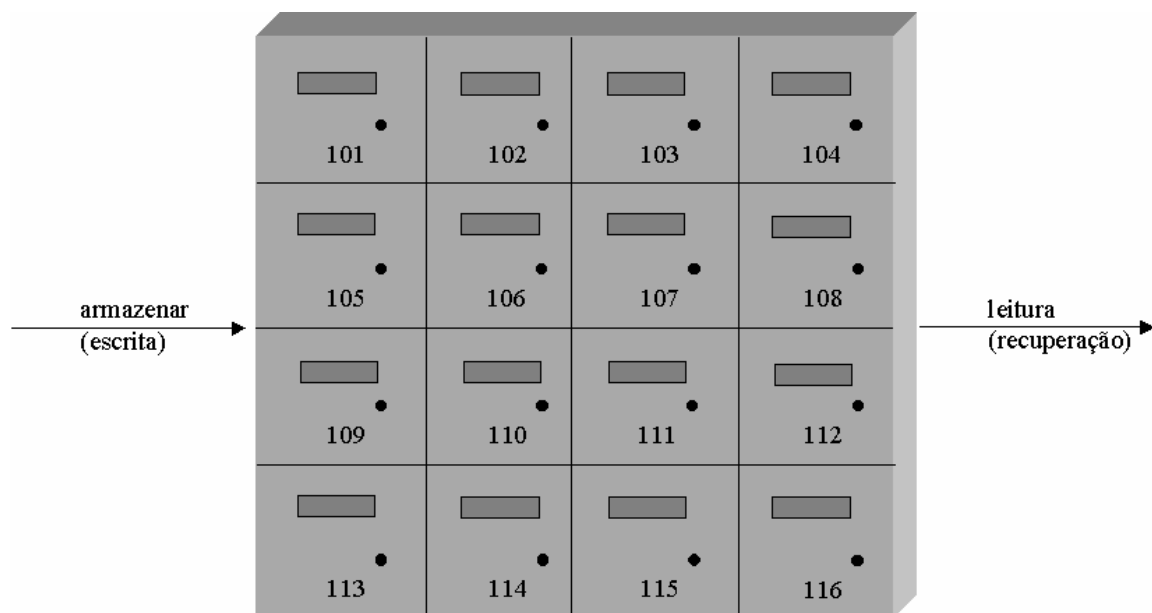
O *hardware* é composto de três grandes partes: CPU, memória e periféricos. Essas partes são os componentes físicos do computador.

Unidade Central de Processamento

A UCP (Unidade Central de Processamento) pode ser vista como o componente mais importante do *hardware*. É na UCP que se encontra o processador central do equipamento.

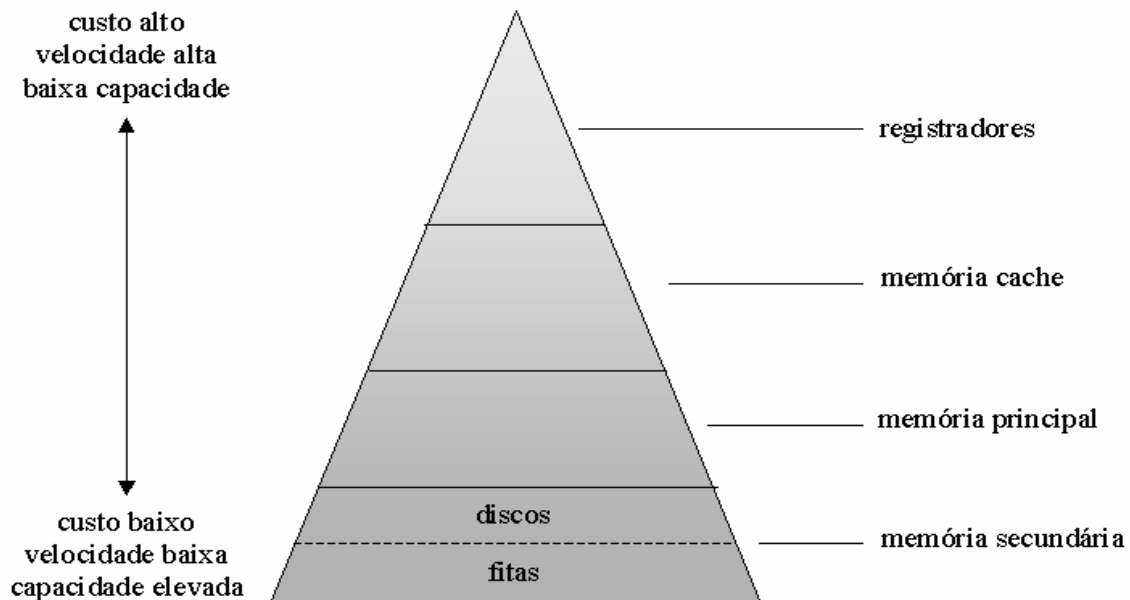
O processador central de um equipamento é a "peça" que comanda todas as suas funções, controla todos os seus componentes, permite acesso de outros equipamentos, realiza as tarefas de lógica e aritmética.

O processador central é composto dos seguintes elementos:

- *Unidade de Controle:* responsável pela verificação e pelo controle das instruções realizadas pela UCP;
- *Unidade de Lógica e Aritmética:* responsável pela realização das operações aritméticas e lógicas;
- *Registrador de Dados:* armazena temporariamente os operandos referentes à instrução que está sendo realizada pela UCP;
- *Registrador de Instruções:* armazena temporariamente a instrução que está sendo realizada pela UCP;
- *Via de Dados (BUS):* permite a conexão da UCP com as demais partes do computador.

Memórias

Conceitualmente, a memória é um componente muito simples: é um "depósito" onde são guardados certos elementos para serem usados quando desejado.

A memória de um computador possui diferentes variedades:

- de velocidade;
- de capacidade de armazenamento;
- de tecnologia de construção;
- de utilidade dentro do sistema.

Para o correto e eficaz funcionamento da manipulação das informações, verifica-se a necessidade de se ter em um mesmo computador diferentes tipos de memória.

Periféricos

Os periféricos são um conjunto de elementos “acessórios” ao fluxo de processamento de dados.

Periféricos de Entrada

- *Teclado*: permite entrada de códigos, caracteres numéricos e alfanuméricos na memória do computador.
- *Mouse*: permite a integração visual entre o usuário e o equipamento; é representado por uma peça que possui uma esfera giratória na parte inferior que, quando em contato com uma superfície plana, reproduz o efeito no monitor de vídeo do equipamento.
- *Scanner*: permite a digitalização de uma imagem pré-impressa; quando utilizado em conjunto com programas específicos chamados OCR (*Optical Caracter Recognition*), praticamente vem dispensar o trabalho de digitação de materiais já impressos.
- *Light Pen* (Caneta Óptica): permite “apontar” com uma espécie de “caneta” o monitor de vídeo de um computador e assim realizar escolhas de opções predefinidas.
- *Leitores de Códigos de Barras*: são dispositivos cujas características são estabelecidas com o propósito de atender às necessidades de leitura de códigos de barras.
- *Joystick*: é atualmente rotulado como um periférico apenas para diversão.

Periféricos de Saída

- *Monitor de Vídeo*: permite visualizar as etapas do processamento de dados, desde a entrada dos dados até a saída das informações.
- *Impressora*: a função desse periférico é perpetuar as informações obtidas pelo processamento por meio da impressão em folhas.
- *Plotter*: conhecido também como traçador gráfico, é um periférico cujo objetivo se assemelha ao da impressora, porém é de uso mais específico: destina-se à confecção de trabalhos gráficos.

1.2. Software

1.2.1. Introdução

Praticamente de nada adianta a existência apenas do *hardware*. O *hardware* tem de ser encarado com um potencial de recursos a serem extraídos na medida das necessidades de seu usuário. Para que seja possível usufruir desse potencial é necessária a parte lógica do computador: o *software*.

O *software* é a parte lógica que dota o equipamento físico de capacidade para realizar todo tipo de trabalho. Por trás do *software* estão as **linguagens de programação**, que são regras básicas que permitem a melhor integração e buscam o ótimo relacionamento entre o *software*, o *hardware* e também o *peopleware*.

1.2.2. Linguagens de Programação

Linguagem de Máquina

- É a linguagem de programação mais próxima da máquina.
- O computador consegue "entender" somente essa linguagem.
- Essa linguagem é a única que consegue enviar instruções ao processados.

Linguagem de Baixo Nível

- É a linguagem de programação ainda bem próxima da máquina, porém com alguns artifícios que a torna mais acessível ao usuário.
- O programador deve conhecer bem, tecnicamente o equipamento que estará programando para extrair dele o mais benefício possível.

Linguagem de Alto Nível

- São linguagens de programação mais acessíveis ao ser humano.
- São constituídas de regras e códigos mais próximos da linguagem do ser humano.

- Não exigem alto grau de conhecimento das características técnicas do equipamento.

1.2.3. Categorias de Software

Software Básico

- O *software* básico é a primeira “interface” com a máquina.
- Essa categoria de *software* realiza o elo de ligação entre a máquina e os demais *softwares*.
- Exemplos: sistemas operacionais, compiladores, interpretadores, etc...

Software de Suporte

- São os programas capazes de administrar as aplicações efetuadas
- Exemplos: *softwares* gerenciadores de rede de computadores, *softwares* que controlam os espaços de memória em disco por usuário, etc...

Software Aplicativo

- Esses *softwares* têm por característica principal estarem voltados para um objetivo previamente definido, porém com menor grau de abrangência.
- Exemplos: programas de editoração de textos, sistemas gerenciadores de banco de dados, planilhas de cálculo, etc...

Software Aplicativo Específico

- São uma especificidade dos *softwares* aplicativos.
- Sua principal característica está nos objetivos estrita e claramente definidos, bem como seu campo de atuação é bem reduzido.
- Exemplos: sistemas de contabilidade, de folha de pagamento, de controle de estoque, etc...

2. Sistemas de Numeração

2.1. Introdução

O homem, através dos tempos, sentiu a necessidade da utilização de sistemas numéricos.

Existem vários sistemas numéricos, dentre os quais se destacam: o sistema decimal, o binário, o octal e o hexadecimal.

O sistema decimal é utilizado por nós no dia-a-dia e é, sem dúvida, o mais importante dos sistemas numéricos. Trata-se de um sistema que possui dez algarismos, com os quais podemos formar qualquer número através da lei de formação. Os outros sistemas, em especial o binário e o hexadecimal, são muito importantes nas áreas de técnicas digitais e informática.

2.2. O Sistema Binário de Numeração

No sistema binário de numeração, existem apenas 2 algarismos: o algarismo 0 (zero) e o algarismo 1 (um). Para representarmos a quantidade zero, utilizamos o algarismo 0, para representarmos a quantidade um utilizamos o algarismo 1. E para representarmos a quantidade dois, se não possuímos o algarismo 2 nesse sistema?

No sistema decimal, não possuímos o algarismo dez e representamos a quantidade de uma dezena utilizando algarismo 1 seguido de algarismo 0. Neste caso, o algarismo 1 significa que temos um grupo de uma dezena e o algarismo 0 nenhuma unidade. No sistema binário, agimos da mesma forma. Para representarmos a quantidade dois, utilizamos o algarismo 1 seguido do algarismo 0.

Na prática, cada dígito binário recebe a denominação de *bit* (***bi**nary digi**t***) e o conjunto de 8 *bits* é denominado de *byte*.

Conversão do sistema binário para o sistema decimal

Utilizando um número decimal qualquer, temos:

$$1524 = 1 \times 1000 + 5 \times 100 + 2 \times 10 + 4$$
$$1 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$$

A regra básica de formação de um número consiste no somatório de cada algarismo correspondente multiplicado pela base elevada por um índice conforme o posicionamento do algarismo no número.

Conversão do sistema decimal para o sistema binário

Para converter um número representado na base 10 para a base 2, tem-se que aplicar um processo para a parte inteira e outro para a parte fracionária.

Para transformar um número inteiro na base 10 para a base 2 utiliza-se o método das divisões sucessivas, que consiste em dividir o número por 2, a seguir divide-se o quociente encontrado por 2 e assim o processo é repetido até que o último quociente seja igual a 1. O número binário será formado pela concatenação do último quociente com os restos das divisões lidos em sentido inverso ao que foram obtidos.

Para transformar um número fracionário da base 10 para a base 2, utiliza-se o método das multiplicações sucessivas, que consistem em multiplicar o número fracionário por 2; deste resultado, a parte inteira será o primeiro dígito do número na base 2. O processo é repetido até que a parte fracionária do último produto seja igual a zero.

2.3. O Sistema Hexadecimal de Numeração

O sistema hexadecimal possui 16 algarismos, sendo sua base igual a 16. Os algarismos são assim enumerados: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F. A letra A representa o algarismo A, que por sua vez representa a quantidade dez. A letra B representa o algarismo B que representa a quantidade onze, e assim sucede até a letra F que representa a quantidade quinze.

Conversão do sistema hexadecimal para o sistema decimal

Para convertermos um número hexadecimal em decimal, utilizaremos o conceito básico de formação de um número.

Conversão do sistema decimal para o sistema binário

O processo é análogo à conversão do sistema decimal para o sistema binário, somente que neste caso, utilizaremos a divisão por 16, pois sendo o sistema hexadecimal, sua base é igual a 16.

Conversão do sistema hexadecimal para o sistema binário

A regra consiste em transformar cada algarismo diretamente no correspondente em binário, respeitando-se o número padrão de *bits* do sistema, sendo o hexadecimal igual a 16 ($2^4 = 16$).

Conversão do sistema binário para o sistema hexadecimal

Neste caso, devemos separar o número em grupos de 4 bits a partir da direita. Em seguida, devemos efetuar a conversão de cada grupo de bits diretamente para o sistema hexadecimal.

3. Desenvolvimento de Programas

3.1. Fases para a solução de um problema:

1. **Definição do problema:** É a descrição e a delimitação adequada do problema a ser resolvido, caracterizando-o de maneira clara e completa.

2. **Desenvolvimento de um algoritmo:** É a descrição, geralmente dividida em etapas, de diversas operações que devem ser realizadas pelo computador para resolver o problema e obter os resultados desejados.

3. **Transcrição do algoritmo para uma linguagem de programação:** É a codificação do algoritmo, obtendo-se um programa equivalente, de acordo com as regras e recursos oferecidos pela linguagem.

4. **Processamento do programa pelo computador:** Seguindo as instruções de um programa denominado compilador, o computador verifica a correção sintática do programa e o traduz para outra linguagem (linguagem de máquina) que, em seguida, passa a controlar o computador através de suas instruções, executando a resolução do problema e obtendo os resultados desejados.

5. **Análise dos resultados:** Os resultados obtidos pelas primeiras execuções de um programa devem ser analisados, pois podem estar comprometidos com erros de lógica.

3.2. Algoritmos

3.2.1. Definição de Algoritmos

A palavra ***algoritmo***, à primeira vista, parece-nos estranha. Embora possua designação desconhecida, fazemos uso constantemente de algoritmos em nosso cotidiano: a maneira como uma pessoa toma banho é um algoritmo.

Definimos algoritmo como a seqüência de passos que visam atingir um objetivo bem definido.

Os algoritmos são utilizados no dia-a-dia para a solução dos mais diversos problemas. Outros algoritmos freqüentemente encontrados são:

- instruções para utilizar um aparelho eletrodoméstico;
- uma receita para preparo de algum prato;
- a maneira como as contas de água, luz e telefone são calculados mensalmente, etc.

São vários os conceitos para algoritmo:

"Um conjunto finito de regras que provê uma seqüência de operações para resolver um tipo de problema específico"
[KNUTH]

"Seqüência ordenada, e não ambígua, de passos que levam à solução de um dado problema"
[TREMBLAY]

"Processo de cálculo, ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema"
[AURÉLIO]

Definição formal de algoritmo: é a descrição de um padrão de comportamento, representado por um repertório finito e bem definido de ações, das quais damos por certo que elas podem ser executadas.

3.2.2. Por que precisamos de algoritmos?

Vejamos o que algumas pessoas importantes para a Ciência da Computação disseram a respeito de algoritmo:

"A noção de algoritmo é básica para toda a programação de computadores".
[KNUTH - Professor da Universidade de Stanford, autor da coleção "The art of computer programming"]

"O conceito central da programação e da ciência da computação é o conceito de algoritmo".
[WIRTH - Professor da Universidade de Zurique, autor de diversos livros na área e responsável pela criação de linguagens de programação como ALGOL, PASCAL e MODULA -2]

A importância do algoritmo está no fato de termos que especificar uma seqüência de passos lógicos para que o computador possa executar uma tarefa qualquer, pois o mesmo por si só não tem vontade própria, faz apenas o que mandamos.

Com uma ferramenta algorítmica, podemos conceber uma solução para um dado problema, independendo de uma linguagem específica e até mesmo do próprio computador.

3.2.3. Característica

Todo algoritmo deve apresentar algumas características básicas:

- ter fim;
- não dar margem à dupla interpretação (não ambíguo);
- capacidade de receber dado(s) de entrada do mundo exterior;
- poder gerar informações de saída para o mundo externo ao do ambiente algoritmo;
- ser efetivo (todas as etapas especificadas no algoritmo devem ser alcançáveis em um tempo finito).

4. Portugol

Durante nosso curso iremos aprender a desenvolver nossos algoritmos em uma pseudo-linguagem conhecida como “Portugol” ou Português Estruturado.

“Portugol” é derivado da aglutinação de Português + Algol. Algol é o nome de uma linguagem de programação estruturada usada no final da década de 50.

4.1. Estrutura Seqüencial

É uma seqüência de comandos, separados por ponto e vírgula. São executados em seqüência linear de cima para baixo.

```
início
   <declaração de variáveis>

   C1;
   C2;
   ...
   Cn;
fim
```

onde C_{is} são comandos, $1 \le i \le n$

4.2. Constantes

Constantes são endereços de memória destinados a armazenar informações fixas, inalteráveis durante a execução do algoritmo.

Conforme seu tipo, a constante é classificada como numérica, literal ou lógica.

Exemplo:

PI

3,141592654

Esta constante irá guardar o valor de PI.

4.3. Variáveis

Uma variável é uma posição de memória, representada por um nome simbólico, atribuído pelo usuário, a qual contém, em um determinado momento um valor de um determinado tipo.

O nome de uma variável é representado por um identificador.

4.3.1. Identificadores

Representam os nomes escolhidos para rotular as variáveis, constantes, novos tipos, procedimentos, funções e conjuntos. Tudo o que pode ser nomeado em um algoritmo deve obedecer às seguintes regras:

1)Devem possuir como 1º caractere uma letra ou sublinhado (_);
2)Ter como demais caracteres letras, números ou sublinhado;
3)Os nomes devem ser formados por caracteres pertencentes ao seguinte conjunto:

- •As letras do alfabeto no padrão inglês maiúsculo e minúsculo:
  abcdefghijklmnopqrstuvxwyz abcdefghijklmnopqrstuvxwyz
- •Os dez algarismos: **0123456789**
- •O sublinhado (Underline): _

4)Ter no máximo 127 caracteres;
5)Não possuir espaços em branco;
6)Os nomes escolhidos devem explicitar seu conteúdo.
7)O identificador não pode ser uma palavra reservada. (veremos adiante)

4.3.2. Tipos de dados

A linguagem Portugol algoritmo possui 4 tipos básicos de dados:

`inteiro`: qualquer número inteiro negativo, nulo ou positivo. *(número sem casas decimais)*
Exemplos : -984563 , 1 , 5 , 0 , 321458

`real`: qualquer número real negativo, nulo ou positivo. *(número pode ter casas decimais)*
Exemplos : -984563 , 1 , 5 , 0 , 321458 , -98,4588 , 45,789

`caracter`: qualquer conjunto de caracteres alfanuméricos. *(letras e números e símbolos especiais)*
Exemplos : “Ana” , “João”, “Rua do ouro” , “123” , “#?*\!&”

`lógico`: tipo especial de variável que armazena apenas os valores `verdadeiro` ou `falso`

4.3.3. Declaração de variáveis

Para que os programas manipulem valores, estes devem ser armazenados em variáveis e para isso, devemos declará-las de acordo com a sintaxe:

```
tipo: identificador ;
```

Exemplos :

```
inteiro: Idade;
caracter: Nome;
real: salario;
lógico: fumante;
```

Se houver mais de uma variável do mesmo tipo deve-se definir de acordo com a sintaxe:

```
tipo: identificador1, identificador2, ..., identificadorn ;
```

Exemplos :

```
inteiro: Idade, num_dependentes ;
caracter: Nome, endereco, rg, estado_civil, nacionalidade ;
real: salario, percentual ;
lógico: fumante;
```

Quando uma variável é declarada, o compilador cria um local na memória, rotulado com o nome simbólico da variável e determina o tipo de valores que ele pode conter.

4.3.4. Declaração de constantes

Devemos declará-las de acordo com a sintaxe:

```
const identificador = valor;
```

Exemplo :

```
const pi = 3.141592654;
```

4.4. Operações Básicas

Na solução da grande maioria dos problemas é necessário que as variáveis tenham seus valores consultados ou alterados. Para isto, devemos definir um conjunto de operadores, sendo eles: aritméticos, lógicos e relacionais.

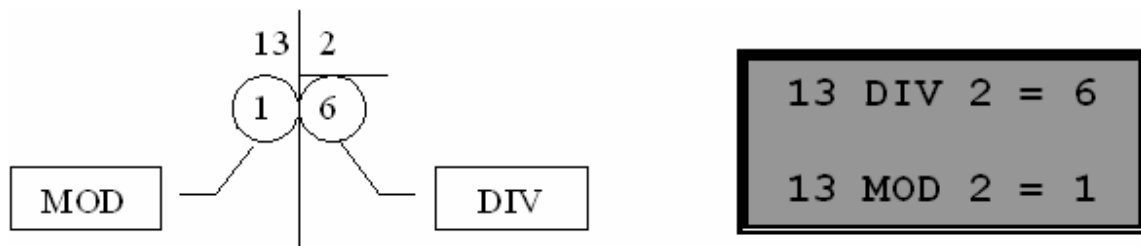
4.4.1. Operadores aritméticos

São usados para representar as operações matemáticas, e obedecem a mesma prioridade da matemática, ou seja, em um comando que aparecem várias operações básicas envolvidas, primeiro é resolvida a POTENCIAÇÃO , depois a MULTIPLICAÇÃO ou a DIVISÃO e por último a SOMA ou a SUBTRAÇÃO.

Operador	Ação
+	Adição
*	Multiplicação
-	Subtração ou inversor do sinal
/	Divisão
**	Exponenciação

Operadores Especiais

`div`: Retorna o valor inteiro que resulta da divisão entre 2 números inteiros.
`mod`: Retorna o resto da divisão entre 2 números inteiros.

Exemplo:

Linearização de Expressões

Para a construção de algoritmos todas as expressões aritméticas devem ser linearizadas, ou seja, colocadas em linhas.

É importante também ressalvar o uso dos operadores correspondentes da aritmética tradicional para a computacional.

Exemplo:

$$\left[\frac{2}{3}+(5-3)\right]+1=$$

Expressão Matemática Tradicional

(2/3+(5-3))+1=

Expressão Computacional

Como pode ser observado no exemplo em expressões computacionais usamos somente parênteses “( )”. Em uma expressão computacional podemos ter parênteses dentro de parênteses.

Exemplos de prioridades:

$$\underbrace{(2+2)/2}_{(4)/2=2} \neq \underbrace{2+2/2}_{2+1=3}$$

4.4.2. Operadores relacionais

São utilizados para comparar variáveis ou expressões, resultando num valor lógico (verdadeiro ou falso), sendo eles:

Operador	Comparação
>	Maior que
<	Menor que
≥	Maior ou igual
≤	Menor ou igual
=	Igual
≠	Diferente

4.4.3. Operadores relacionais

São utilizados para avaliar expressões lógicas, sendo eles:

Operador	Ação
não	Inverte o valor, de verdadeiro para falso e vice-versa
e	Retorna verdadeiro se ambas as partes forem verdadeiras
ou	Basta que um valor seja verdadeiro para retornar verdadeiro

Tabela Verdade:

P	Q	não P	P e Q	P ou Q
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

Exemplos:

(2 + 5 ≥ 7) e (3 ≠ 3) = falso
(10+8/2=14) ou (30 - 7<10) = verdadeiro
(2+5 ≥ 7) e (não(3 ≠ 3)) = verdadeiro

4.4.3. Funções

Uma função é um instrumento (sub–algoritmo) que tem como objetivo retornar um valor ou uma informação. A chamada de uma função é feita através da citação do seu nome seguido opcionalmente de seu argumento inicial entre parênteses.

As funções podem ser predefinidas pela linguagem ou criadas pelo programador de acordo com o seu interesse.

Bibliotecas de funções armazenam um conjunto de funções que podem ser usadas pelos programas.

Função	Resultado	Exemplo	Tipo de Resultado
Sen (X)	Sendo X um ângulo em radianos. A função Sen (X) retorna o seno de X em radianos.	Sen (90) = 0,893997	Real
Cos (X)	Sendo X um ângulo em radianos. A função Cos (X) retorna o Cosseno de X em radianos.	Cos (30) = 0,154251	Real
Tg (X)	Sendo X um ângulo em radianos. A função Tg (X) retorna a Tangente de X em radianos.	Tg (60) = 0,32004	Real
Arctg (X)	Sendo X um ângulo em radianos. A função Arctg (X) retorna o Arco Tangente de X em radianos.	Arctg (180)= 1,565241	Real
Arredonda(X)	Sendo X uma variável ou um valor numérico. A função Arredonda (X) retorna o valor inteiro mais próximo de X.	Arredonda (8,5) = 9	Real
Sinal (X)	Sendo X uma variável ou um valor numérico. A função sinal (X) retorna +1 para x positivo; -1 para X negativo; e 0 para x nulo.	Sinal (-9)= -1 Sinal (9) = 1 Sinal (0) = 0	Inteiro
Int (X)	Sendo X uma variável ou um valor numérico. A função Int (X) retorna a parte inteira de X, desprezando a parte decimal de X,sem arredondar.	Int (-8,899)= -8,0	Real
Raiz (X)	Sendo X uma variável ou um valor numérico. A função Raiz (X) retorna a raiz quadrada de X	Raiz(400) = 20	Real
Abs (X)	Sendo X uma variável ou um valor numérico. A função Abs (X) retorna o valor absoluto de X ou seja, X sem o sinal.	Abs (-98)=98	Real ou Inteiro

As funções acima são as mais comuns e importantes para nosso desenvolvimento lógico, entretanto, cada linguagem possui suas funções próprias.

4.4.5. Prioridade de operadores

Durante a execução de uma expressão que envolve vários operadores, é necessário a existência de prioridades, caso contrário poderemos obter valores que não representam o resultado esperado.

A maioria das linguagens de programação utiliza as seguintes prioridades de operadores:

1)Efetuar operações embutidas em parênteses "mais internos"
2)Efetuar funções
3)Efetuar exponenciação
4)Efetuar multiplicação e/ou divisão e/ou `div` e/ou `mod`
5)Efetuar adição e/ou subtração
6)Operadores relacionais
7)Operadores lógicos na ordem a seguir:
  7.1) `não`
  7.2). `e`
  7.3) `ou`

4.5. Comandos de Entrada e Saída

Em um algoritmo é preciso representar a troca de informações que ocorrerá entre o mundo da máquina e o nosso mundo, para isso, devemos utilizar comandos de entrada e saída, sendo que, em nível de algoritmo esses comandos representam apenas a entrada e a saída da informação, independe do dispositivo utilizado (teclado, discos, impressora, monitor,...), mas, sabemos que nas linguagens de programação essa independência não existe, ou seja, nas linguagens de programação temos comandos específicos para cada tipo de unidade de entrada/saída.

4.5.1. Comando de entrada de dados

O comando de entrada de dados é utilizado para armazenar em uma variável um valor que se encontra em uma unidade de entrada de dados (usualmente o teclado). Existe apenas um comando simples para a entrada de dados, o comando `leia`:

```
leia(variavel);
```

onde: `leia` é uma palavra reservada
`variável` representa a variável onde será armazenado o valor proveniente da unidade de entrada, qualquer que seja ela.

Para ler e armazenar mais de um valor ao mesmo tempo deve-se usar o comando leia da seguinte forma:

```
leia(variavel1, variavel2, ..., variaveln);
```

Neste exemplo o primeiro valor digitado será armazenado na $variavel_1$, o segundo valor digitado será armazenado na $variavel_2$ e assim por diante.

4.5.2. Comando de saída de dados

O comando de saída de dados é utilizado para exibir em uma unidade de saída (usualmente o monitor) uma mensagem e/ou dados do algoritmo. Existe apenas um comando simples para a exibição de resultados, o comando imprima.

- Se quisermos que seja exibido o conteúdo de uma variável devemos usar o comando imprima da seguinte forma:

```
imprima(variavel);
```

onde: imprima é uma palavra reservada
variavel representa a variável cujo conteúdo será exibido na unidade de saída.

- Se houver mais de uma variável que queiramos exibir seu conteúdo devemos usar o comando imprima da seguinte forma:

```
imprima(variavel1, variável2, ..., variáveln);
```

onde: as variáveis deverão vir separadas por vírgula.

- Se quisermos exibir um texto qualquer:

```
imprima("texto qualquer");
```

onde: o texto que será exibido sempre deve vir entre aspas.

- Se quisermos exibir um texto e o conteúdo de uma variável juntos, devemos usar o comando imprima da seguinte forma:

```
imprima("texto" , variavel);
```

- Se quisermos exibir o resultado do processamento de uma expressão lógica ou aritmética, devemos usar o comando imprima da seguinte forma:

```
imprima(12,5 – 3);
```

- Se quisermos exibir um texto, o conteúdo de uma variável e o resultado do processamento de uma expressão lógica juntos, devemos usar o comando `imprima` da seguinte forma:

```
imprima("texto" , variável , expressão);
```

4.6. Comando de Atribuição

Esta instrução é um comando para "colocar" um valor na área de uma variável. E pode ocorrer das seguintes formas:

- Atribuição de constante:

```
variavel ← valor;
```

onde: `variavel` é o nome da variável que receberá um dado `valor`, este valor deverá ser compatível ao tipo da variável.

- Atribuição de variável:

$variavel_1 \leftarrow variavel_2;$

onde: $variavel_1$ é o nome da variável que receberá o conteúdo da $variavel_2$. As variáveis devem ser do mesmo tipo.

- Atribuição de expressão:

```
variavel ← expressao;
```

onde: `variavel` é o nome da variável que receberá um valor através de: uma expressão aritmética (envolvendo um cálculo), expressão lógica (envolvendo uma comparação) ou expressão literal (envolvendo um processamento de texto). Após a expressão ser avaliada ou "calculada", o valor obtido é atribuído à variável.

4.7. Estrutura Condicional

Uma das tarefas fundamentais de qualquer algoritmo é decidir o que deve ser executado a seguir. A estrutura condicional permite determinar qual é a ação a ser tomada com base no resultado de um teste lógico. A estrutura condicional pode ser formada por uma alternativa simples ou composta.

4.7.1. Alternativa Simples

Na alternativa simples se a expressão de teste (`condição`) for verdadeira, os comandos que estão dentro da cláusula `então` são executados; do contrário, a execução do algoritmo continua na linha seguinte ao `fim se`.

```
se (condição)
     então
        C1;
        C2;
        ...
        Cn;
fim se
```

onde `condição` é qualquer expressão cujo resultado seja verdadeiro ou falso
C_{is} são comandos, $1 \leq i \leq n$

4.7.2. Alternativa Composta

Na alternativa composta se a expressão de teste (`condição`) for verdadeira, os comandos que estão dentro da cláusula `então` são executados; do contrário, os comandos que estão dentro da cláusula `senão` são executados.

```
se (condição)
     então
        α1;
        α2;
        ...
        αn;
     senão
        β1;
        β2;
        ...
        βn;
fim se
```

onde `condição` é qualquer expressão cujo resultado seja verdadeiro ou falso
α_{is} são comandos, $1 \leq i \leq n$
β_{js} são comandos, $1 \leq j \leq n$

4.8. Estrutura de Repetição

A estrutura de repetição é útil sempre que uma ou mais instruções (comandos) devam ser repetidas enquanto uma certa condição estiver sendo satisfeita.

Na estrutura de repetição `enquanto`, se a expressão de teste (`condição`) for verdadeira, as instruções que estão dentro do `enquanto` são executadas uma vez e a expressão de teste é avaliada novamente. Este ciclo de teste e execução é repetido até que a expressão de teste se torne falsa, então a execução do algoritmo continua na instrução seguinte ao `fim enquanto`.

```
enquanto (condição) faça
      C1;
      C2;
      ...
      Cn;
fim enquanto
```

onde `condição` é qualquer expressão cujo resultado seja verdadeiro ou falso
C_{is} são comandos, $1 \leq i \leq n$

4.9. Regras Práticas para a Construção de Algoritmos Legíveis

1. Procure incorporar comentários no algoritmo.
2. Escolha nomes de variáveis que sejam significativos.
3. Grife todas as palavras-chave do algoritmo.
4. Procure alinhar os comandos de acordo com o nível a que pertençam.

4.10. Exemplos de Algoritmos

1. Escrever um algoritmo para ler 4 notas bimestrais de um aluno. Calcular e imprimir a média aritmética das 4 notas.

```
início
      inteiro: nota1, nota2, nota3, nota4;
      real: media;

      imprima("Digite 4 notas: ");
      leia(nota1, nota2, nota3, nota4);
      media ← (nota1+nota2+nota3+nota4)/4;
      imprima("Media das notas: ", media);
fim
```

2. Escrever um algoritmo para ler o raio de uma circunferência, calcular e imprimir sua área.

```
início
     const pi = 3.1415;
     real: raio, área;

     imprima("Entre com o valor do raio da circunferencia:");
     leia(raio);
     área ← pi * sqr(raio);
     imprima("Área da circuferencia: ", área);
fim
```

3. Escrever um algoritmo para ler dois números e imprimir se esses números são iguais ou não.

```
início
     inteiro: numero1, numero2;

     imprima("Entre com dois numeros:");
     leia(numero1, numero2);
     se (numero1=numero2)
          então imprima("Números iguais");
          senão imprima("Números diferentes");
     fim se
fim
```

4. Escrever um algoritmo para imprimir os cem primeiros números em ordem crescente.

```
início
     inteiro: contador;

     contador ← 1;
     enquanto (contador ≤ 100) faça
          imprima(contador, " ");
          contador ← contador + 1;
     fim enquanto
fim
```